Dockerized Full-Stack Deployment on AWS EC2

Documentation

1.Overview

- This project is a Packers & Movers web application deployed on AWS EC2 using Docker containers for the React.js frontend, Spring Boot backend, and MySQL database.
- The goal is to demonstrate a production-like DevOps workflow: provisioning infrastructure, containerizing services, configuring networking and security, and validating end-to-end data flow.

2. Architecture

- **AWS EC2:** Single Amazon Linux instance hosting all Docker containers.
- **Docker network:** Custom bridge network that allows the backend and MySQL containers to communicate using container names.
- **Frontend-Container:** Serves the UI on port 5000 and calls backend APIs using the EC2 public IP and backend port.
- **Backend-Container:** Exposes REST APIs on port 8080 and connects to the MySQL container using JDBC configuration in `application.properties(filename)`.
- **MySQL-container:** Runs the application database with custom database, user, and password configured via environment variables.

3. Prerequisites

- AWS account with permission to create EC2 instances and security groups.
- EC2 instance (Amazon Linux or similar) with:
  → Inbound rules for ports 22 (SSH), 5000 (frontend), 8080 (backend), 3306 (MySQL).
- Installed on EC2:
  → Docker engine
  → GIT
- GitHub repositories for:
  → docker-assign-frontend
  → docker-assign-backend

4. Setup and Deployment

4.1 Launch EC2 and security group:

1. Login into aws management and go to **EC2 → Instances → Launch instances.**
2. Choose an Amazon Linux AMI and instance type (for example, `t2.small`).

3. Configure the security group to allow inbound:

- SSH (22) from your IP
- HTTP/Custom TCP 5000 from 0.0.0.0/0
- HTTP/Custom TCP 8080 from 0.0.0.0/0
- MySQL/Aurora 3306 from 0.0.0.0/0 (or locked down to your IP for better security).

4. Launch the instance and connect via SSH using your key pair.

4.2 Install Docker and prepare environment:

- **Installation of Docker:**
  - → Sudo yum update –y
  - → Sudo yum install docker –y
  - → Sudo systemctl start docker
  - → Sudo usermod –a –G docker ec2-user
- **Create a dedicated Docker network:**
  - → Sudo docker network create app-net
  - → Sudo docker network ls

4.3 Run MySQL container:

- **Launching of mysql-container:**

```
→ Sudo docker run -d --name mysql-container \
   --network app-net \
   -e MYSQL_ROOT_PASSWORD=<22Qqwwee> \
   -e MYSQL_DATABASE=upendar-db \
   -e MYSQL_USER=upendar-user \
   -e MYSQL_PASSWORD=<upendar-pass> \
   -p 3306:3306 mysql:latest
```

- **Verify of database container creation:**
  → Sudo docker ls

4.4 Configure backend-container:

1.Clone of backend repository:

→ Git clone https://github.com/upendarbandam/docker-assign-backend.git backend-repo
→ Cd backend-repo

2.Open src/main/resources/application.properties and set database configuration:

→ spring.datasource.url=jdbc:mysql://mysql-container:3306/upendar-db
spring.datasource.username=upendar-user
spring.datasource.password=<upendar-pass>
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.
MySQL8Dialect

3.Build the backend Docker image:
→ Sudo docker build –t backend-image .
→ Sudo docker images

4. **Launching of backend-container:**

→ Cd ../
→ Sudo docker run –d –name backend-container –network app-net –p 8080:8080 backend-image
→ Sudo logs backend-container (this needs to show Tomcat running on port 8080)

4.5 Configure and launch of frontend-container:

1.Clone the frontend repository:

- → git clone https://github.com/upendarbandam/docker-assign-frontend.git frontend-repo
- → Cd frontend-repo

2.In the frontend code (for example `src/services/index.js` or equivalent), update the API base URL to use the EC2 public IP and backend port:

- → const API_BASE_URL = http://98.95.0.71:8080;

3.Build the frontend docker image:

- → Sudo docker build –t frontend-image .
- → Sudo docker images

4. Launching of frontend-container:

- → Sudo docker run –d –-name frontend-container –p 5000:5000 frontend-image
- → Sudo Docker ps
- → Sudo logs frontend-container

  (Expect message that frontend/BFF server is running on port 5000)

5. Validation and Troubleshooting

5.1 Validate containers and networking:

- List running containers:
  → sudo docker ps
  (Expect mysql-container, backend-container, frontend-container)
- Inspect network to confirm containers are attached:
  → Sudo docker network inspect app-net

5.2 Validate database data:

1.Enter the MySQL container:

- → sudo docker exec -it mysql-container /bin/bash
- → mysql -u upendar-user –p

2.Run basic SQL checks:

→ Show databases ;

→ Use upendar-db ;

→ Show tables ;

→ Select * from customers ;

(You should see records created from the React UI)

5.3 Common issues:

- **Frontend cannot reach backend:**
  - → Check API base URL in frontend code matches `http://<EC2_PUBLIC_IP>:8080`.
  - → Confirm security group allows inbound traffic on port 8080.

- **Backend cannot reach MySQL:**
  - → Confirm backend and MySQL containers are on `app-net`.
  - → Check `spring.datasource.url` uses `mysql-container:3306`, not `localhost`.
- **Port already in use:**
  - → Ensure no other process is bound to 5000, 8080, or 3306; stop old containers or processes if needed.

6. Future Improvements

- Automate build and deployment using a CI/CD pipeline (GitHub Actions, Jenkins, or AWS CodePipeline).
- Move from a single EC2 instance to AWS ECS/EKS for container orchestration and better scalability
- Use AWS Secrets Manager or Systems Manager Parameter Store for database credentials instead of plain environment variables.
- Add monitoring and alerting using Amazon CloudWatch metrics, logs, and alarms.